

7-3-00

A

UTILITY PATENT APPLICATION TRANSMITTAL

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Attorney Docket No. 042390.P8130First Named Inventor or Application Identifier Jayashankar BharadwajExpress Mail Label No. EL591668285US

JC863 U.S. PTO

09/608616

06/30/00

ADDRESS TO: Assistant Commissioner for Patents
Box Patent Application
Washington, D. C. 20231

APPLICATION ELEMENTS

See MPEP chapter 600 concerning utility patent application contents.

1. Fee Transmittal Form
(Submit an original, and a duplicate for fee processing)
2. X Specification (Total Pages 29)
(preferred arrangement set forth below)
 - Descriptive Title of the Invention
 - Cross References to Related Applications
 - Statement Regarding Fed sponsored R & D
 - Reference to Microfiche Appendix
 - Background of the Invention
 - Brief Summary of the Invention
 - Brief Description of the Drawings (if filed)
 - Detailed Description
 - Claims
 - Abstract of the Disclosure
3. X Drawings(s) (35 USC 113) (Total Sheets 14)
4. Oath or Declaration/Power of Attorney (Total Pages)
 - a. Newly Executed (Original or Copy)
 - b. Copy from a Prior Application (37 CFR 1.63(d))
(for Continuation/Divisional with Box 17 completed) (**Note Box 5 below**)
 - i. DELETIONS OF INVENTOR(S) Signed statement attached deleting inventor(s) named in the prior application, see 37 CFR 1.63(d)(2) and 1.33(b).
5. Incorporation By Reference (useable if Box 4b is checked)
The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.
6. Microfiche Computer Program (Appendix)
7. Nucleotide and/or Amino Acid Sequence Submission
(if applicable, all necessary)
 - a. Computer Readable Copy
 - b. Paper Copy (identical to computer copy)
 - c. Statement verifying identity of above copies

ACCOMPANYING APPLICATION PARTS

8. _____ Assignment Papers (cover sheet & documents(s))
9. _____ 37 CFR 3.73(b) Statement (where there is an assignee)
10. _____ English Translation Document (if applicable)
11. _____ a. Information Disclosure Statement (IDS)/PTO-1449
_____ b. Copies of IDS Citations
12. _____ Preliminary Amendment
13. X _____ Return Receipt Postcard (MPEP 503) (Should be specifically itemized)
14. _____ a. Small Entity Statement(s)
_____ b. Statement filed in prior application, Status still proper and desired
15. _____ Certified Copy of Priority Document(s) (if foreign priority is claimed)
16. X _____ Other: Declaration/Power of Attorney (Unsigned)

17. If a **CONTINUING APPLICATION**, check appropriate box and supply the requisite information:
 _____ Continuation _____ Divisional _____ Continuation-in-part (CIP)
 of prior application No: _____

18. Correspondence Address

_____ Customer Number or Bar Code Label _____
 (Insert Customer No. or Attach Bar Code Label here)
 or
X _____ Correspondence Address Below
 NAME Michael A. DeSanctis Reg. No. 39,957 6/30/2000
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 Wilshire Boulevard 7th Floor, Los Angeles, California 90025
(303) 740-1980 Telephone (303) 740-6962 Facsimile

EXPRESS MAIL CERTIFICATE OF MAILING

"Express Mail" mailing label number: EL591668285US
 I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed to the Assistant Commissioner for Patents, Washington, DC 20231.

June 30, 2000
 Date of Deposit
April Worley
 Name of Person Mailing Correspondence
April Worley 6/30/00
 Signature Date

Docket No: 042390.P8130
 Express Mail Label: EL591668285US

UNITED STATES PATENT APPLICATION
FOR
USER TRANSPARENT CONTINUOUS COMPILATION

INVENTORS:

JAYASHANKAR BHARADWAJ
a citizen of the United States,
residing at 19553 Brockton Lane Saratoga, CA 95070

RAVI NARAYANASWAMY
a citizen of India,
residing at 1336 Muench Ct. San Jose, CA 95131

PREPARED BY:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP
12400 WILSHIRE BOULEVARD
SEVENTH FLOOR
LOS ANGELES, CA 90025-1026
(303) 740-1980

EXPRESS MAIL CERTIFICATE OF MAILING

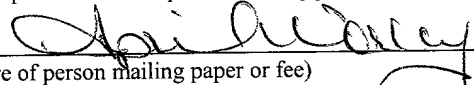
“Express Mail” mailing label number: EL591668285US

Date of Deposit: June 30, 2000

I hereby certify that I am causing this paper or fee to be deposited with the United States Postal Service
“Express Mail Post Office to Addressee” service on the date indicated above and that this paper or fee has
been addressed to the Commissioner of Patents and Trademarks, Washington, D. C. 20231

April M. Worley

(Typed or printed name of person mailing paper or fee)


(Signature of person mailing paper or fee)

June 30, 2000
(Date signed)

USER TRANSPARENT CONTINUOUS COMPILATION

FIELD OF THE INVENTION

This invention relates to compilers in general, and more specifically to a
5 transparent continuous compiler that provides an optimization of a software application
customized to a user's system and usage.

BACKGROUND OF THE INVENTION

Most often, software does not make the best use of the hardware on which it runs.
10 This is due to many reasons. For one, most compiler optimizations are machine
dependent. When generating an executable for general distribution software vendors must
make decisions about what kind of machine the user is likely to be using. A vendor may
choose to optimize his code for the newest processor available while another may opt for
another approach. Beside the processor type, other elements of the users system such as
15 the cache size and bus speed affect how the machine performs and therefore relate to how
code should be optimized to best perform on that system. Additionally, a user's hardware
configuration may change over time due to hardware upgrades. This further complicates
the choices to be made in generating optimized code.

Another factor effecting how software should best be optimized is the manner in
20 which the software will be used. That is, the functions utilized and the data set upon
which the program operates effects how the code should be optimized. Again, software
vendors are left with the task of determining the most likely uses and type of data set that

will be used. As with optimizations directed to specific hardware, these choices to not optimum for all possible users of the software.

To generate an optimized executable, software vendors can utilize profile guided optimization. Historically, profile guided optimization in the compiler has been done by
5 inserting instrumentation into the code to be optimized, compiling the code and then executing the instrumented code on a representative machine with a representative data set. The instrumentation provides feedback that allows the software vendor to make adjustments to the code to reach optimum performance on the test machine. Current systems to enable profile collection and usage in the compiler are tedious and
10 consequently usage of profile feedback among software vendors is very low. A software vendor may not have access to representative program inputs. As explained above, the software vendor usually has to choose a single target machine configuration when optimizing the binary that is shipped. This choice is often non-optimal. Hence, to generate a high performance executable, knowledge of accurate usage profiles and the
15 target machine is imperative. Ideally code should be optimized for an individual user and allow for changes over time due to hardware upgrades and changes in usage.

BRIEF DESCRIPTION OF THE DRAWINGS

The appended claims set forth the features of the invention with particularity. The invention, together with its advantages, may be best understood from the following detailed description taken in conjunction with the accompanying drawings of which:

5 Figure 1 is a block diagram of a typical computer system upon which one embodiment of the present invention may be implemented;

Figure 2 is a block diagram illustrating distribution of a software product according to one embodiment of the present invention;

10 Figure 3 is a block diagram illustrating generation of a software product according to one embodiment of the present invention;

Figure 4 is a flowchart illustrating a high level overview of transparent continuous compilation according to one embodiment of the present invention;

Figure 5 is a flowchart illustrating an installation process according to one embodiment of the present invention;

15 Figure 6 is a flowchart illustrating program execution processing according to one embodiment of the present invention;

Figure 7 is a flowchart illustrating a recompilation process according to one embodiment of the present invention;

20 Figure 8 is a block diagram illustrating a system for transparent continuous compilation according to one embodiment of the present invention;

Figure 9 is a block diagram illustrating a compiler annotation according to one embodiment of the present invention;

Figure 10 is a flow chart illustrated annotation creation according to one embodiment of the present invention;

Figure 11 is a flow chart illustrating annotation duplication according to one embodiment of the present invention;

5 Figure 12 is a flow chart illustrating annotation deletion according to one embodiment of the present invention;

Figure 13 is a flow chart illustrating annotation merging according to one embodiment of the present invention; and

10 Figure 14 is a flowchart illustrating annotation branch inversion according to one embodiment of the present invention.

DETAILED DESCRIPTION

In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art that the present invention may be practiced without some of these specific details. In other instances, well-known structures and devices are shown in block diagram form.

The present invention includes various steps, which will be described below. The steps of the present invention may be performed by hardware components or may be embodied in machine-executable instructions, which may be used to cause a general-purpose or special-purpose processor or logic circuits programmed with the instructions to perform the steps. Alternatively, the steps may be performed by a combination of hardware and software.

The present invention may be provided as a computer program product which may include a machine-readable medium having stored thereon instructions which may be used to program a computer (or other electronic devices) to perform a process according to the present invention. The machine-readable medium may include, but is not limited to, floppy diskettes, optical disks, CD-ROMs, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, magnet or optical cards, flash memory, or other type of media / machine-readable medium suitable for storing electronic instructions. Moreover, the present invention may also be downloaded as a computer program product, wherein the program may be transferred from a remote computer to a requesting computer by way of data signals embodied in a carrier wave or other propagation

medium via a communication link (e.g., a modem or network connection).

Figure 1 is a block diagram of a typical computer system upon which one embodiment of the present invention may be implemented. Computer system 100 comprises a bus or other communication means 101 for communicating information, and a processing means such as processor 102 coupled with bus 101 for processing information. Computer system 100 further comprises a random access memory (RAM) or other dynamic storage device 104 (referred to as main memory), coupled to bus 101 for storing information and instructions to be executed by processor 102. Main memory 104 also may be used for storing temporary variables or other intermediate information during execution of instructions by processor 102. Computer system 100 also comprises a read only memory (ROM) and/or other static storage device 106 coupled to bus 101 for storing static information and instructions for processor 102.

A data storage device 107 such as a magnetic disk or optical disc and its corresponding drive may also be coupled to computer system 100 for storing information and instructions. Computer system 100 can also be coupled via bus 101 to a display device 121, such as a cathode ray tube (CRT) or Liquid Crystal Display (LCD), for displaying information to an end user. Typically, an alphanumeric input device 122, including alphanumeric and other keys, may be coupled to bus 101 for communicating information and/or command selections to processor 102. Another type of user input device is cursor control 123, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 102 and for controlling cursor movement on display 121.

A communication device 125 is also coupled to bus 101. The communication device 125 may include a modem, a network interface card, or other well known interface devices, such as those used for coupling to Ethernet, token ring, or other types of physical attachment for purposes of providing a communication link to support a local or wide area network, for example. In this manner, the computer system 100 may be coupled to a number of clients and/or servers via a conventional network infrastructure, such as a company's Intranet and/or the Internet, for example.

It is appreciated that a lesser or more equipped computer system than the example described above may be desirable for certain implementations. Therefore, the configuration of computer system 100 will vary from implementation to implementation depending upon numerous factors, such as price constraints, performance requirements, technological improvements, and/or other circumstances.

It should be noted that, while the steps described herein may be performed under the control of a programmed processor, such as processor 102, in alternative embodiments, the steps may be fully or partially implemented by any programmable or hardcoded logic, such as Field Programmable Gate Arrays (FPGAs), TTL logic, or Application Specific Integrated Circuits (ASICs), for example. Additionally, the method of the present invention may be performed by any combination of programmed general purpose computer components and/or custom hardware components. Therefore, nothing disclosed herein should be construed as limiting the present invention to a particular embodiment wherein the recited steps are performed by a specific combination of hardware components.

As described above, the software vendor usually has to choose a single target machine configuration when optimizing the binary that is shipped. This choice is often non-optimal. Hence, to generate a high performance executable, accurate profiles of usage and the target machine are imperative. Ideally code should be optimized for an individual user and allow for changes over time due to hardware upgrades and changes in usage. One solution is to distribute the program in an intermediate level representation which can be read by a variety of machines along with a continuous compiler and a runtime monitor which will be used for generating a customized executable, optimized for the user's machine.

Figure 2 is a block diagram illustrating distribution of a software product according to one embodiment of the present invention. According to this embodiment, the software vendor 200 will distribute to the user 215 a package 210 consisting of several parts 220-230. The package 210 includes an intermediate representation (IR) 220 of the application, a continuous compiler 225, and a runtime monitor 230. The intermediate representation 220 is a binary representation of the original source generated by the software vendor that can be read by the continuous compiler on the user's machine. The compiler 225 and runtime monitor 230 are used during compilation and execution of an executable generated from the intermediate representation 220. These elements will be described in greater detail below.

In alternative embodiments it is contemplated that the software vendor may distribute only an intermediate representation 220 of the source code. The compiler 225

and runtime monitor 230 may be distributed separately. In another embodiment, the compiler 225 and runtime monitor 230 may be part of an operating system.

To generate an intermediate representation, the compiler is broken into two parts, a front end used by the software vendor that takes high level source code as input and produces a high level intermediate representation of the program, and a back-end or continuous compiler run on the user's machine that takes IR files and a profile database, if available, as input and produces objects or executables. The continuous compiler also attaches annotations to the IR files as described below. Generic, non-user specific compiler optimizations can be made by the front-end by the software vendor and all the profile dependant and machine dependant optimizations are part of the continuous compiler. After successful validation, the software vendor would distribute the high level intermediate representation of the program.

Figure 3 is a block diagram illustrating generation of a software product according to one embodiment of the present invention. Such a process could be implemented by software vendors distributing software packages suitable for use by various embodiments of the present invention. A source code file 305 written in a high level language such as C is transformed by a compiler 310 into an intermediate representation file 315. This compiler 310 also generates and inserts annotations into the intermediate representation 315. The purpose and function of these annotations will be described in greater detail below. The intermediate representation 315 is then transformed into an executable binary 325 by a continuous compiler 320 similar to the one that will be installed on the users machine.

This executable binary 325 can then be validated 330. The validation process 330 can provide feedback 335 which can be used to modify the source code 305.

The continuous compiler generates a lot of extra code in the executable to perform many safety tests. These tests can include checks for use of uninitialized
5 variables, checks for bad pointers before dereferences, array bounds checks prior to making array references, etc. After validation, the source code 305 can then be debugged and optimized and recompiled 310 to generate a new intermediate representation 315 which can then be distributed to users 340. In this manner a high quality, debugged intermediate representation of the original source code can be distributed that includes
10 some generic optimizations. Further user specific optimizations will be made on the user's machine.

Figure 4 is a flowchart illustrating a high level overview of transparent continuous compilation according to one embodiment of the present invention. This process would be performed on the user's machine after distribution of the software package generated
15 as described above. Several steps are described in general terms with reference to this figure. These steps, such as installing, executing, and recompiling, will be described in greater detail below with reference to figures 5-7.

After the software package has been installed 405, the newly generated executable can be run 410. The performance of the executable on the user's machine is monitored
20 410 profile data is generated. This profile data is compared to past profile data 415. If the profile data has changed in an amount greater than a predetermined threshold amount

415, the system will wait for the CPU to become idle 420. Once the user's machine

becomes idle 420, the intermediate representation can be recompiled 425 using the newly generated profile data to optimize the executable.

Figure 5 is a flowchart illustrating an installation process according to one embodiment of the present invention. Assuming a continuous compiler and runtime monitor are distributed with the software package as indicated in Figure 2, the installation process first checks if a continuous compiler has already been installed 505. If no compiler has yet been installed 505, a compiler is then installed on the user's machine 510. Similarly, if no runtime monitor has yet been installed 515, a runtime monitor is installed on the user's machine 520. The intermediate representation of the software is copied to the target machine 525. Next, a profile database is build for the user's machine 525. This initial profile database includes details of the hardware configuration of the users machine. The intermediate representation can then be compiled 535 using the profile database to generate a customized executable for the user's machine. The details of this compilation are substantially similar to those of the recompilation described with reference to figure 7 below.

In alternative embodiments of the present invention, the continuous compiler and run time monitor may be distributed separate from applications or may be part of an operating system. In such case, the only functions which need to be performed are to copy the intermediate representation to the target machine 525, build an initial profile database or read an existing profile database 530, and compile the intermediate representation 535.

Figure 6 is a flowchart illustrating program execution processing according to one embodiment of the present invention. First, the executable version of the application program is started 605. While the executable is running, the process is sampled at a controlled interval 610. That is, profile data is collected. While the CPU is busy 615, execution of the application program and profile collection continues. When the CPU becomes idle 615, binary level profiles and high level intermediate representation profiles are generated 625.

Figure 7 is a flowchart illustrating a recompilation process according to one embodiment of the present invention. If inter-compilation optimization is possible 705, the compiler optimizations are customized 710. These customizations can include loop unrolling, pipelining, function in-lining etc. If profile optimization is possible 715, customizations for the user's environment are made 720. These customizations can include preloading data to avoid cache misses, branch prediction adjustments etc. Finally, once all customizations have been made, the intermediate representation is again compiled 725 thereby generating a new executable which is customized to the current configuration and usage of the user's hardware and application.

Figure 8 is a block diagram illustrating a system for transparent continuous compilation according to one embodiment of the present invention. As explained previously, an installation process 810 copies an intermediate representation (IR) 815 to the target machine. This installation process may also copy a continuous compiler 820 and a runtime monitor 840 to the machine. Alternatively, the continuous compiler 820

and runtime monitor 840 may already be part of the operating system 800. Installation 810 also generates an initial profile database 830 containing information about the user's hardware configuration. The continuous compiler 820 uses the intermediate representation and profile database 830 to generate an executable version 825 of the intermediate representation 815. When the executable 825 is run 835 the runtime monitor 840 collects information from the hardware and operating system 800 for offline profile analysis 845. The offline profile analysis 845 generates data for the profile database 830. If the data being stored in the profile database 830 shows a change from the previous conditions that exceeds a predetermined threshold, recompilation is triggered. When the CPU next becomes idle, the continuous compiler 820 again uses the intermediate representation 815 and profile database 830 to generate an executable 825 that is now customized to the current profile data 830. The transparency layer 805 allows the user, through the operating system 800, to interact with the current version of the executable 825 without needing to keep track of recompilations.

Software installation would trigger an initial compilation of this IR by the continuous compiler for the observed target microprocessor platform at the user site. As the user uses the program, information is collected on its run-time characteristics. These include characteristics of the machine such as processor type and cache configuration so that we may detect changes due to machine upgrades etc. Hardware performance monitors are sampled to derive binary level program profiles. Profile collection is done by sampling the execution of the user's process at a controlled rate so that the runtime overhead to the user process is not noticeable (1-2%). Collected samples are processed

during CPU idle time in two steps: 1) generation of binary level profiles, from analysis of the collected samples, and annotations generated in the compilation process for flow graph construction and other purposes; and 2) derivation of profiles at the high level intermediate language used for recompilation, from the binary level profiles generated in step 1.

In order to perform optimizations of an executable, it is necessary to be able to relate locations in the executable to the profile database and the intermediate representation. To do so, the present invention uses annotations. Information on branches, on the number of times a block is executed, and on loads causing cache misses is required. In order to collect this information, tag instructions are placed in the code. These tags allow unique identification of instructions through the compiler and analyzer. Additional information is placed in these tags to form the annotations.

These annotations map each binary level instruction to a source level token, and also describe how the binary level instruction evolved from its corresponding high level instruction in the prior compilation. Using these annotations we can synthesize the IR profiles from binary level profiles. Also, in subsequent recompilation when an IR instruction is broken into multiple instructions either through lowering or optimization we can apportion these profiles appropriately between derived operations. For example, a branch that is duplicated through unrolling may have different probabilities and mispredict rates on the duplicated instructions. If this information was captured in the binary level profiles we can use it to reconstruct this information.

Another novel aspect of the annotation feedback scheme enables communication

between phases of the compiler across compilations. This helps reduce phase ordering problems in the compiler. Annotations are used to record events or other relevant information by optimization phases during a compilation. In subsequent compilations these annotations may be consumed by the same or other optimization phases.

5 Figure 9 is a block diagram illustrating a compiler annotation according to one embodiment of the present invention. As indicated, the annotation ar action node 900 contains several pieces of data. First is a major ID 905. This major ID 905 is made up of information based on the source file, line and column number. The action ID 910 is a unique ID to capture a particular action on a particular instruction. The previous actions
10 915 and next actions 920 are pointers to the previous and next action nodes in the list. Actions 925 is an indication of the action that is represented by this node. For example, DELETED, CREATED, etc. Phase 930 refers to the compiler optimization phase in which the node was created. Feedback data 935 includes data collected during program execution. This data can include number of branches 945, load/store information 950,
15 branch probability 955, cache miss indication 960 etc.

According to one embodiment of the present invention, annotations can be implemented as a linked list of data structures. Handling of linked lists is routine in the art but is described with reference to figures 10-14 as it relates to annotations.

Figure 10 is a flow chart illustrated annotation creation according to one
20 embodiment of the present invention. First a new action node is created 1005. The new action node is assigned a major ID from a precomputed ID 1010. Next, a new action number is assigned to the new node 1015. The new nodes previous action pointer is set to

NULL 1020 and the compiler phase in which the node was created is marked 1025.

Finally, the action is marked as CREATED 1030.

Figure 11 is a flow chart illustrating annotation duplication according to one embodiment of the present invention. First, two new action nodes are created 1105. The major ID is copied from the action node of the instruction being copied 1110 and a new action number is assigned to the two new nodes 1115. The previous action pointers of the two new nodes are set to the action node being copied 1120 and the compiler phase in which the node was duplicated is marked 1125. Finally, the action is marked as DUPLICATED. 1130.

Figure 12 is a flow chart illustrating annotation deletion according to one embodiment of the present invention. First, a new action node is created 1205. The major ID is copied to the new node from the action node of the instruction being deleted 1210 and a new action number is assigned 1215. The previous action pointer of the new node is set to the action node of the instruction being deleted 1220 and the compiler phase in which the node was deleted is marked 1225. Finally, the action is marked as DELETED 1230.

Figure 13 is a flow chart illustrating annotation merging according to one embodiment of the present invention. First, a new action node is created 1305. The major ID from one of the previous action nodes of the instructions being merged is copied to the new action node 1310. A new action number is assigned to the new node 1315. Next, the previous actions pointer of the new node is set to a list of action nodes of the instructions being merged. The new node is added to the next actions list of the previous nodes 1325

and the compiler phase in which the node was merged is marked. Finally, the action is marked as MERGED 1335.

Figure 14 is a flowchart illustrating annotation branch inversion according to one embodiment of the present invention. First, a new action node is created 1405. The major ID from the action node of the branch being inverted is copied to the new action node 1310. A new action number is assigned to the new node 1315. Next, the previous actions pointer of the new node is set to the action node of the branch being inverted and the compiler phase in which the node was merged is marked. Finally, the action is marked as BRANCH_INVERTED 1335.

10

CLAIMS

What is claimed is:

- 1 1. A method comprising:
2 installing a program onto a target machine;
3 executing the program; and
4 responsive to a change in profile data collected while the program executes which
5 exceeds a predetermined threshold, recompiling the program while the
6 target machine is idle.
- 1 2. The method of claim of claim 1, wherein said installing further comprises:
2 installing a continuous compiler;

- 3 installing a runtime monitor;
- 4 copying an intermediate representation of a source file to the target machine;
- 5 building a profile database; and
- 6 compiling the intermediate representation to create an executable file.
- 1 3. The method of claim 1, wherein said executing further comprises:
- 2 running an executable version of the program;
- 3 collecting samples of process information at a controlled rate; and
- 4 while the target machine is idle, generating binary level and high level profiles.
- 1 4. The method of claim 1, wherein recompiling further comprises customizing
- 2 compiler optimizations based on profile data generated during program execution.
- 1 5. The method of claim 4, wherein said customizing compiler optimizations is
- 2 performed using annotations in a high level representation of an executable
- 3 program which relate portions of the executable to the high level representation.
- 1 6. The method of claim 6, wherein creating said annotations comprises:
- 2 creating a new action node;
- 3 assigning to the new action node a major ID from a precomputed ID;
- 4 assigning a new action number to the new action node;
- 5 setting a previous action node pointer of the new action node to NULL;
- 6 marking a compiler phase in which the new node was created; and
- 7 marking an action of the new node as created.
- 1 7. The method of claim 6, wherein duplicating an annotation comprises:
- 2 creating two new action nodes;

3 copying a major ID to the new action nodes from an action node of instructions
4 being copied;
5 assigning new action numbers to the two new nodes;
6 setting previous action node pointers of the new nodes to the action node being
7 copied;
8 marking a compiler phase in which the nodes was duplicated; and
9 marking an action of the new nodes as duplicated.

1 8. The method of claim 6, wherein deleting an annotation comprises:
2 creating a new action node;
3 copying a major ID from an action node of an instruction being deleted to the new
4 action node;
5 assigning a new action number to the new action node;
6 setting a previous action pointer in the new action node to the action node of the
7 instruction being deleted;
8 marking a compiler phase in which the node was deleted; and
9 marking an action of the deleted node as deleted.

1 9. The method of claim 6, wherein merging of annotations comprises:
2 creating a new action node;
3 copying a major ID from a previous action node of instructions being merged;
4 assigning a new action number to the new action node;
5 setting a previous action pointer of the new node to a list of nodes of the
6 instruction being merged;

7 adding the new action to a next actions pointer list of previous actions;
8 marking a compiler phase in which the node was merged;
9 marking an action of the new node as merged.

1 10. The method of claim 6, wherein annotation branch inversion comprises:
2 creating a new action node;
3 copying a major ID from a branch instruction being inverted;
4 assigning a new action number to the new node;
5 setting a previous action pointer of the new node to a node of a branch being
6 inverted;
7 marking a compiler phase in which the inversion occurred; and
8 marking the branch as inverted.

1 11. A computer readable medium having stored thereon data representing sequences
2 of instructions, the sequences of instructions which, when executed by a
3 processor, cause the processor to perform the steps of:
4 installing a program onto a target machine;
5 executing the program; and
6 responsive to a change in profile data collected while the program executes which
7 exceeds a predetermined threshold, recompiling the program while the
8 target machine is idle.

1 12. The computer readable medium of claim of claim 11, wherein said installing
2 further comprises:
3 installing a continuous compiler;

- 4 installing a runtime monitor;
- 5 copying an intermediate representation of a source file to the target machine;
- 6 building a profile database; and
- 7 compiling the intermediate representation to create an executable file.
- 1 13. The computer readable medium of claim 11, wherein said executing further
- 2 comprises:
- 3 running an executable version of the program;
- 4 collecting samples of process information at a controlled rate; and
- 5 while the target machine is idle, generating binary level and high level profiles.
- 1 14. The computer readable medium of claim 11, wherein recompiling further
- 2 comprises customizing compiler optimizations based on profile data generated
- 3 during program execution.
- 1 15. The computer readable medium of claim 14, wherein said customizing compiler
- 2 optimizations is performed using annotations in a high level representation of an
- 3 executable program which relate portions of the executable to the high level
- 4 representation.
- 1 16. The computer readable medium of claim 16, wherein creating said annotations
- 2 comprises:
- 3 creating a new action node;
- 4 assigning to the new action node a major ID from a precomputed ID;
- 5 assigning a new action number to the new action node;
- 6 setting a previous action node pointer of the new action node to NULL;

7 marking a compiler phase in which the new node was created; and
8 marking an action of the new node as created.

1 17. The computer readable medium of claim 16, wherein duplicating an annotation
2 comprises:

3 creating two new action nodes;

4 copying a major ID to the new action nodes from an action node of instructions
5 being copied;

6 assigning new action numbers to the two new nodes;

7 setting previous action node pointers of the new nodes to the action node being
8 copied;

9 marking a compiler phase in which the nodes was duplicated; and

10 marking an action of the new nodes as duplicated.

1 18. The computer readable medium of claim 16, wherein deleting an annotation
2 comprises:

3 creating a new action node;

4 copying a major ID from an action node of an instruction being deleted to the new
5 action node;

6 assigning a new action number to the new action node;

7 setting a previous action pointer in the new action node to the action node of the
8 instruction being deleted;

9 marking a compiler phase in which the node was deleted; and

10 marking an action of the deleted node as deleted.

1 19. The computer readable medium of claim 16, wherein merging of annotations
2 comprises:
3 creating a new action node;
4 copying a major ID from a previous action node of instructions being merged;
5 assigning a new action number to the new action node;
6 setting a previous action pointer of the new node to a list of nodes of the
7 instruction being merged;
8 adding the new action to a next actions pointer list of previous actions;
9 marking a compiler phase in which the node was merged;
10 marking an action of the new node as merged.

1 20. The computer readable medium of claim 16, wherein annotation branch inversion
2 comprises:
3 creating a new action node;
4 copying a major ID from a branch instruction being inverted;
5 assigning a new action number to the new node;
6 setting a previous action pointer of the new node to a node of a branch being
7 inverted;
8 marking a compiler phase in which the inversion occurred; and
9 marking the branch as inverted.

1 21. A system comprising:
2 a storage device having stored therein a routine for transparent continuous
3 compilation;

4 a processor coupled to the storage device which when executing the routine:
5 installs a program onto a target machine;
6 executes the program; and
7 responsive to a change in profile data collected while the program executes which
8 exceeds a predetermined threshold, recompiles the program while the
9 target machine is idle.

1 22. The system of claim of claim 21, wherein said installation further comprises:
2 installing a continuous compiler;
3 installing a runtime monitor;
4 copying an intermediate representation of a source file to the target machine;
5 building a profile database; and
6 compiling the intermediate representation to create an executable file.

1 23. The system of claim 21, wherein said execution further comprises:
2 running an executable version of the program;
3 collecting samples of process information at a controlled rate; and
4 while the target machine is idle, generating binary level and high level profiles.

1 24. The system of claim 21, wherein recompilation further comprises customizing
2 compiler optimizations based on profile data generated during program execution.

1 25. The system of claim 24, wherein said customizing compiler optimizations is
2 performed using annotations in a high level representation of an executable
3 program which relate portions of the executable to the high level representation.

1 26. The system of claim 26, wherein creating said annotations comprises:

2 creating a new action node;
3 assigning to the new action node a major ID from a precomputed ID;
4 assigning a new action number to the new action node;
5 setting a previous action node pointer of the new action node to NULL;
6 marking a compiler phase in which the new node was created; and
7 marking an action of the new node as created.

1 27. The system of claim 26, wherein duplicating an annotation comprises:

2 creating two new action nodes;
3 copying a major ID to the new action nodes from an action node of instructions
4 being copied;
5 assigning new action numbers to the two new nodes;
6 setting previous action node pointers of the new nodes to the action node being
7 copied;
8 marking a compiler phase in which the nodes was duplicated; and
9 marking an action of the new nodes as duplicated.

1 28. The system of claim 26, wherein deleting an annotation comprises:

2 creating a new action node;
3 copying a major ID from an action node of an instruction being deleted to the new
4 action node;
5 assigning a new action number to the new action node;
6 setting a previous action pointer in the new action node to the action node of the
7 instruction being deleted;

8 marking a compiler phase in which the node was deleted; and
9 marking an action of the deleted node as deleted.

1 29. The system of claim 26, wherein merging of annotations comprises:
2 creating a new action node;
3 copying a major ID from a previous action node of instructions being merged;
4 assigning a new action number to the new action node;
5 setting a previous action pointer of the new node to a list of nodes of the
6 instruction being merged;
7 adding the new action to a next actions pointer list of previous actions;
8 marking a compiler phase in which the node was merged;
9 marking an action of the new node as merged.

1 30. The system of claim 26, wherein annotation branch inversion comprises:
2 creating a new action node;
3 copying a major ID from a branch instruction being inverted;
4 assigning a new action number to the new node;
5 setting a previous action pointer of the new node to a node of a branch being
6 inverted;
7 marking a compiler phase in which the inversion occurred; and
8 marking the branch as inverted.

ABSTRACT

According to the invention, systems, apparatus and methods are disclosed for installing a program onto a target machine, executing the program, and responsive to a change in profile data collected while the program executes which exceeds a
5 predetermined threshold, recompiling the program while the target machine is idle.

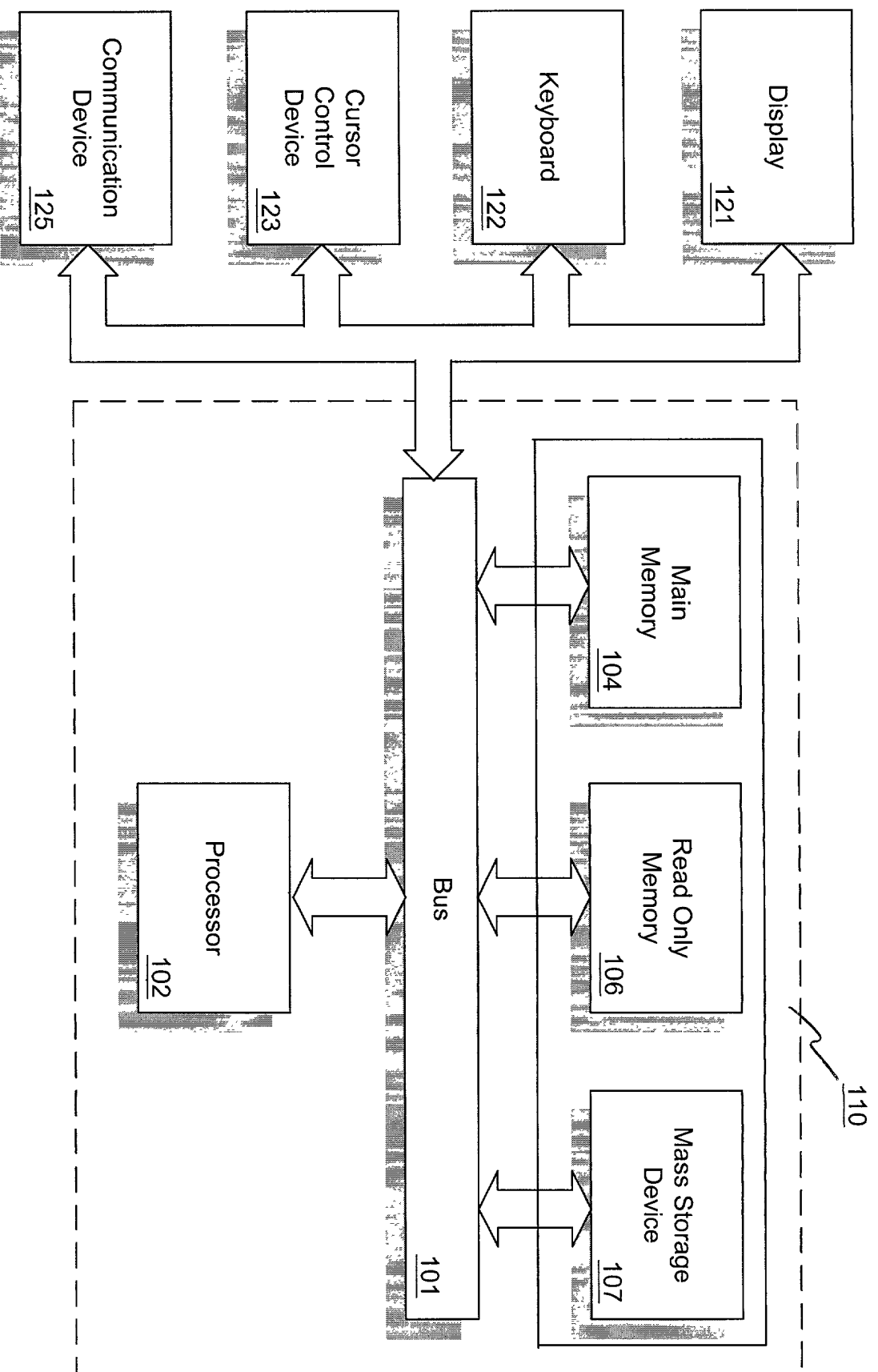


Figure 1

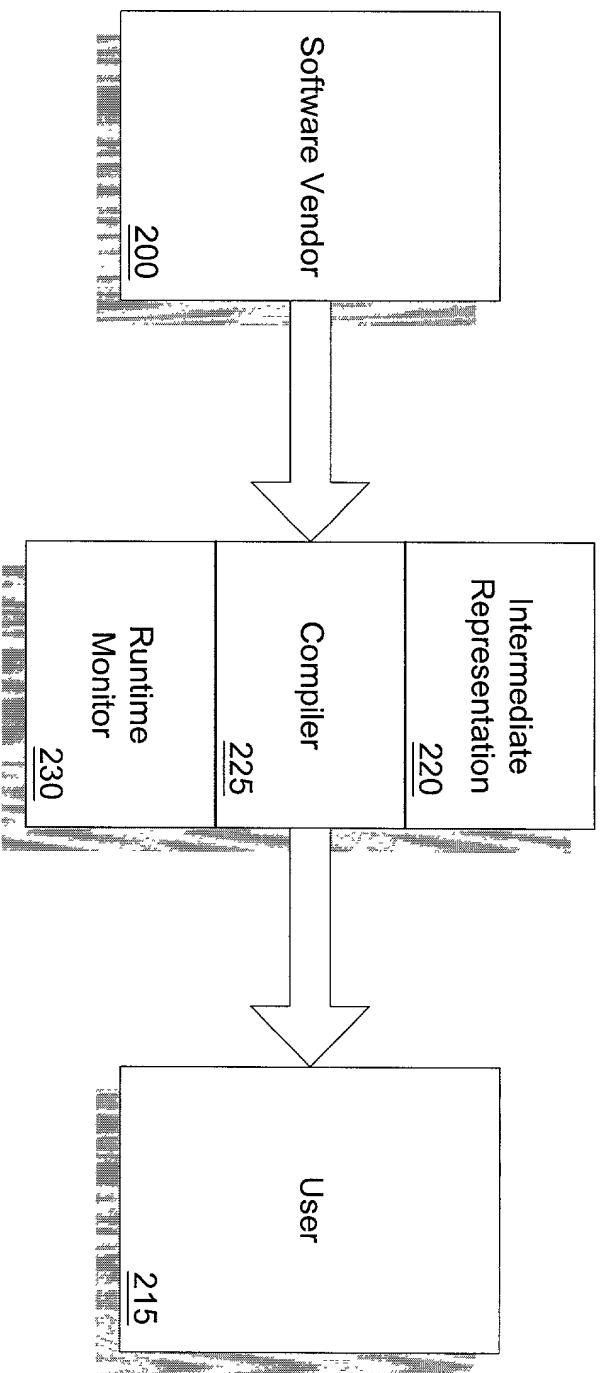


Figure 2

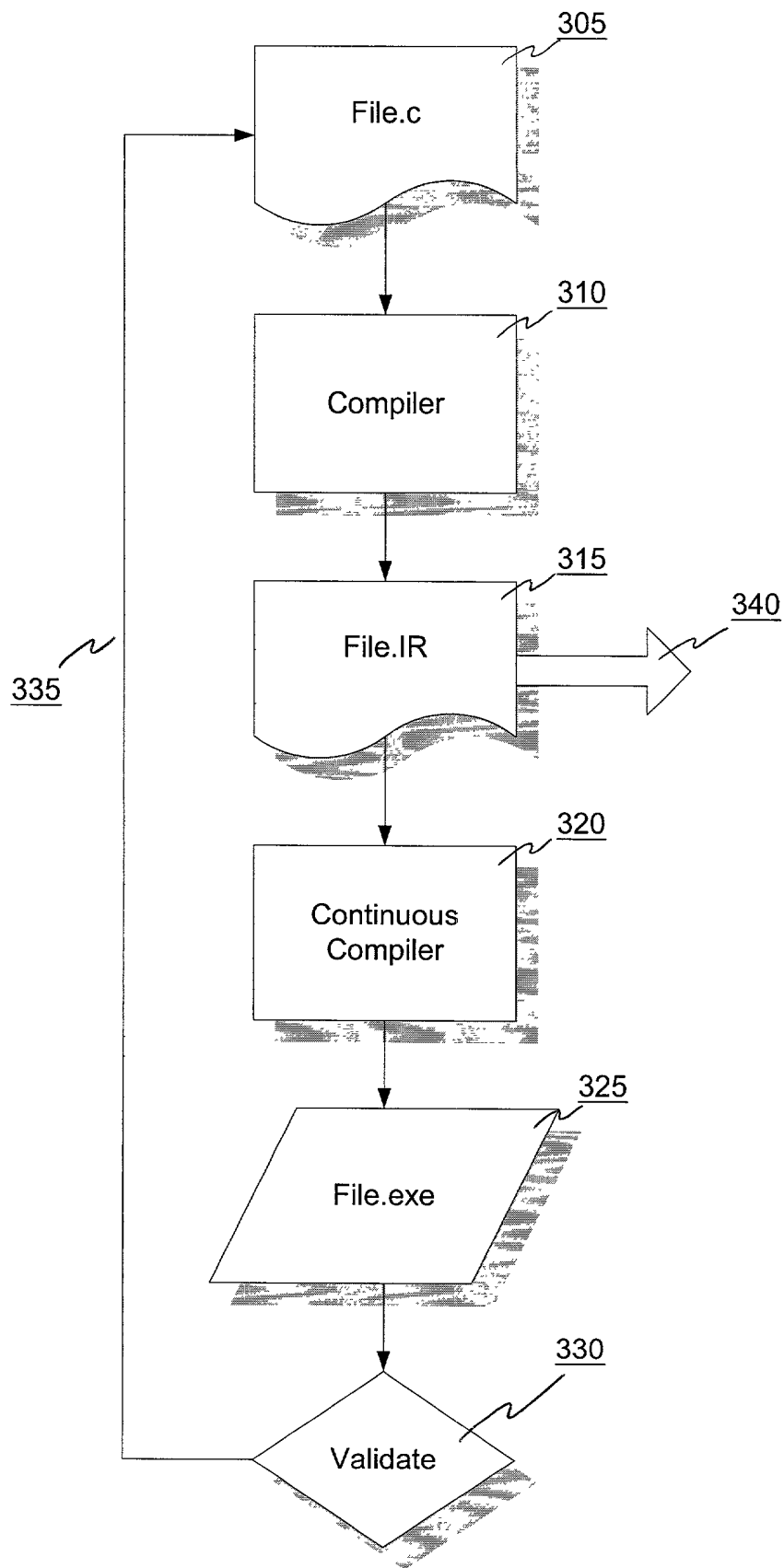


Figure 3

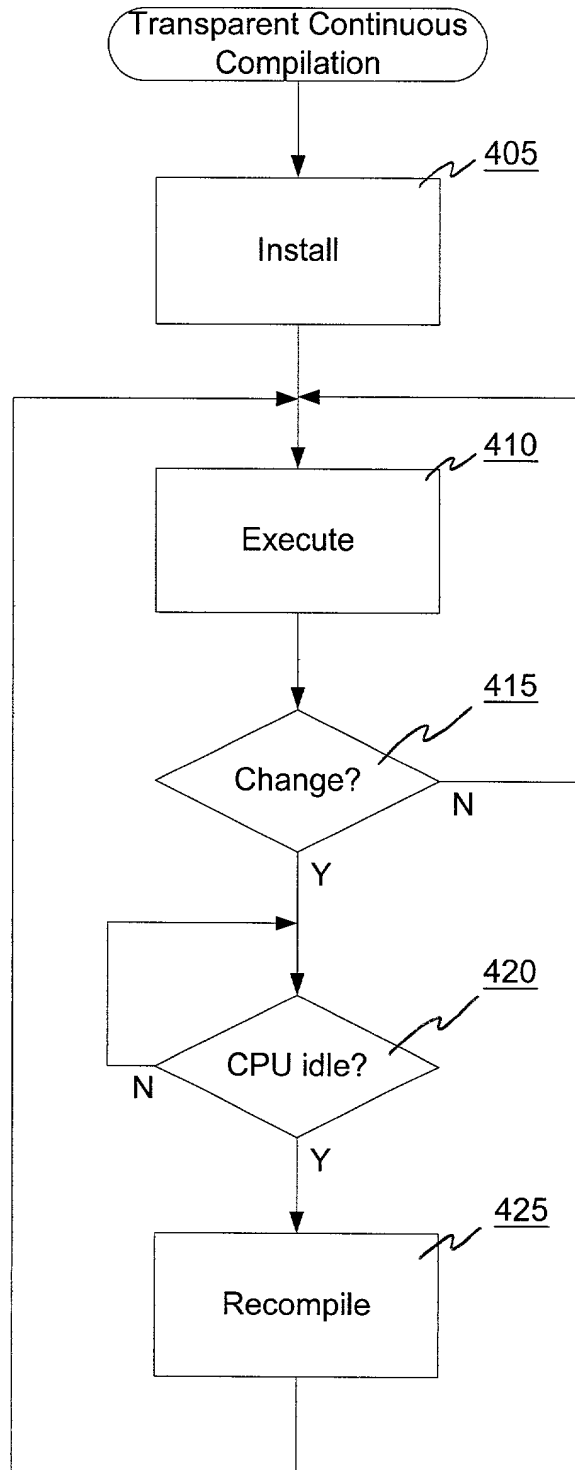


Figure 4

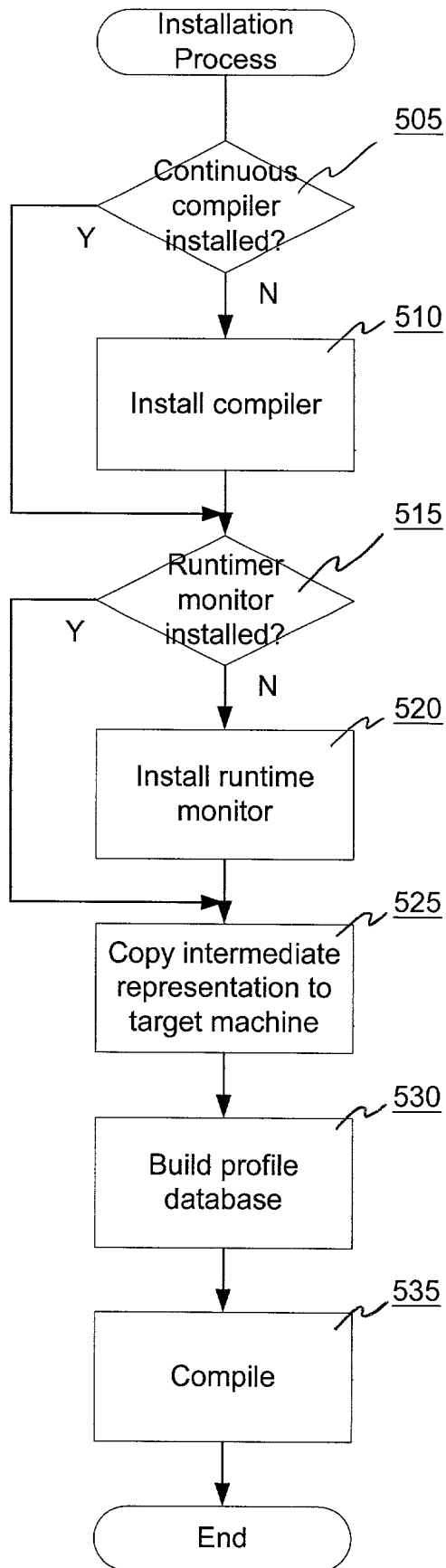


Figure 5

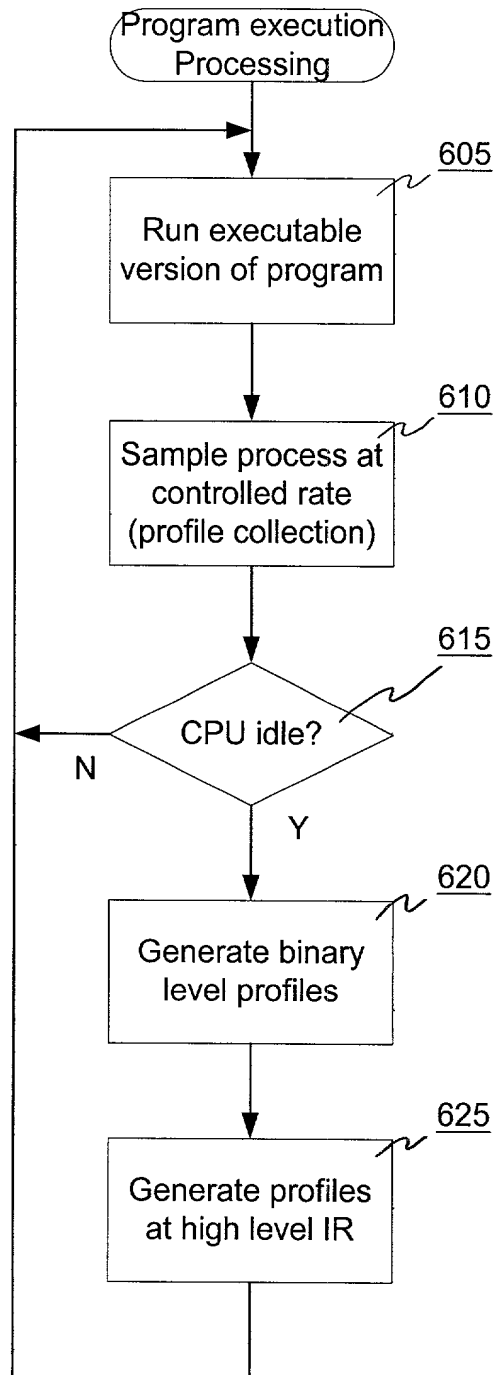


Figure 6

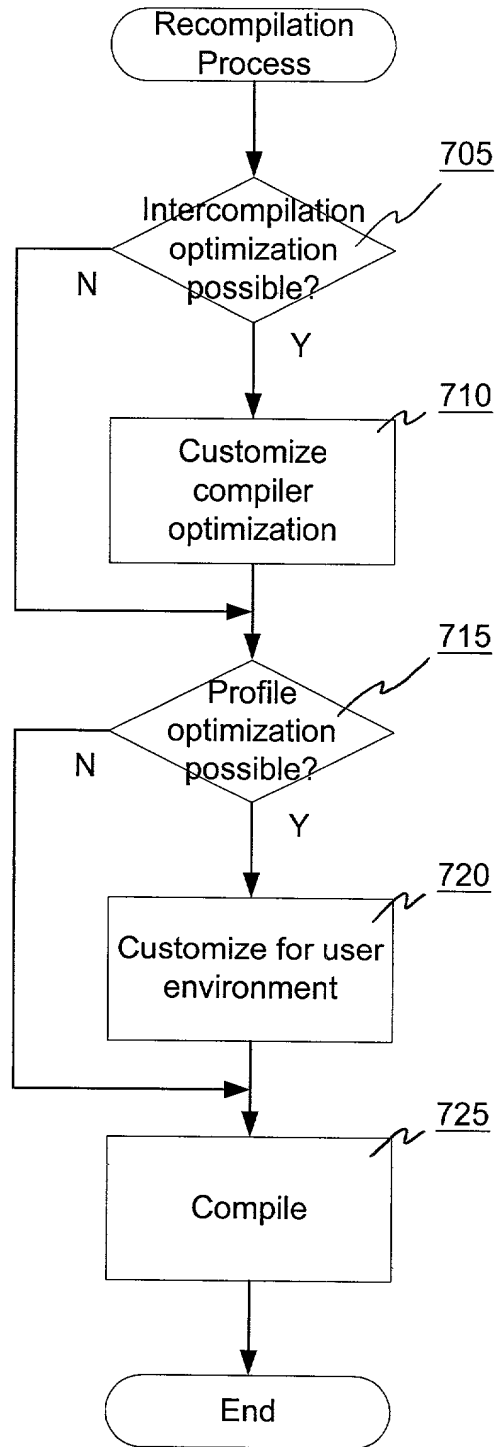


Figure 7

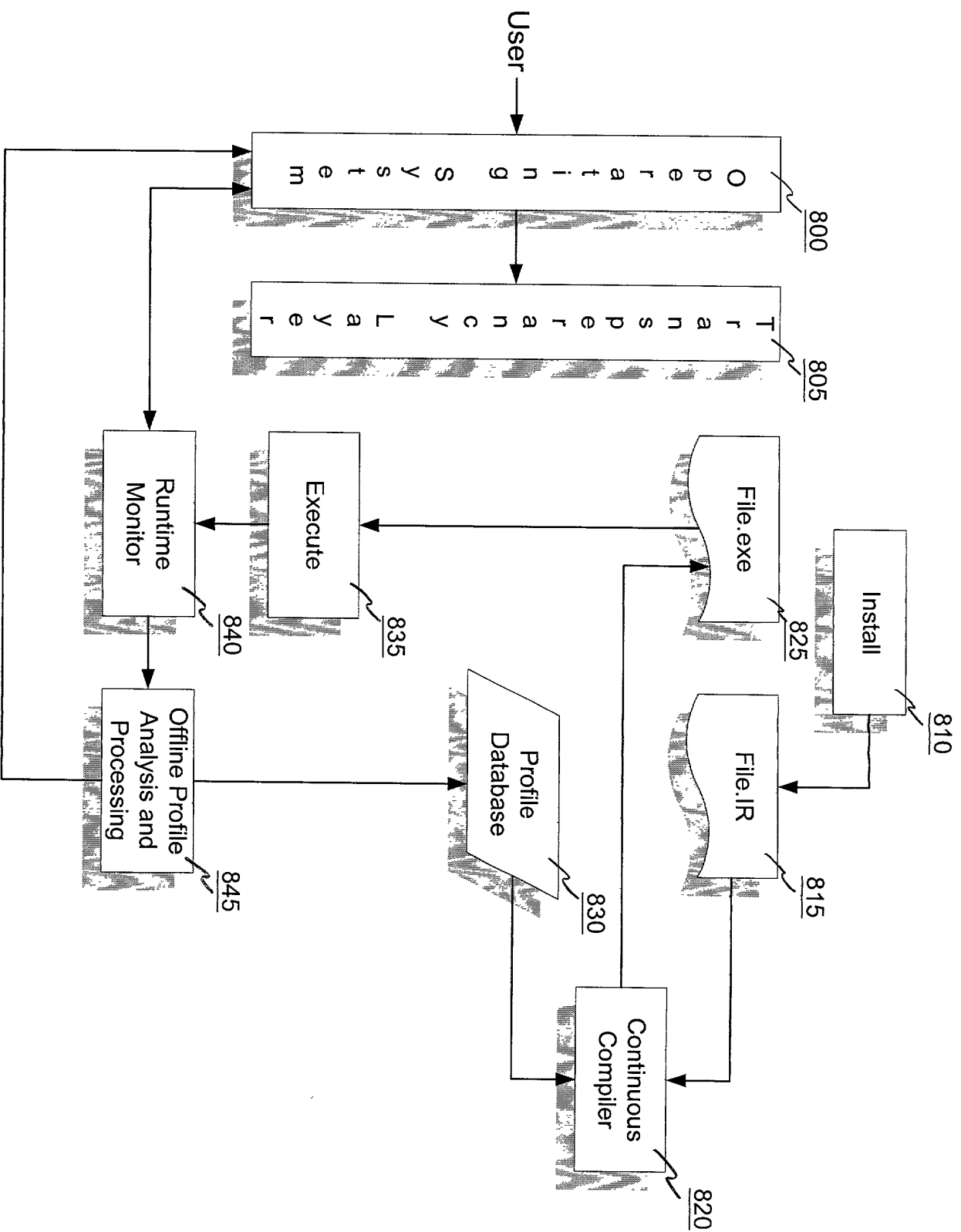


Figure 8

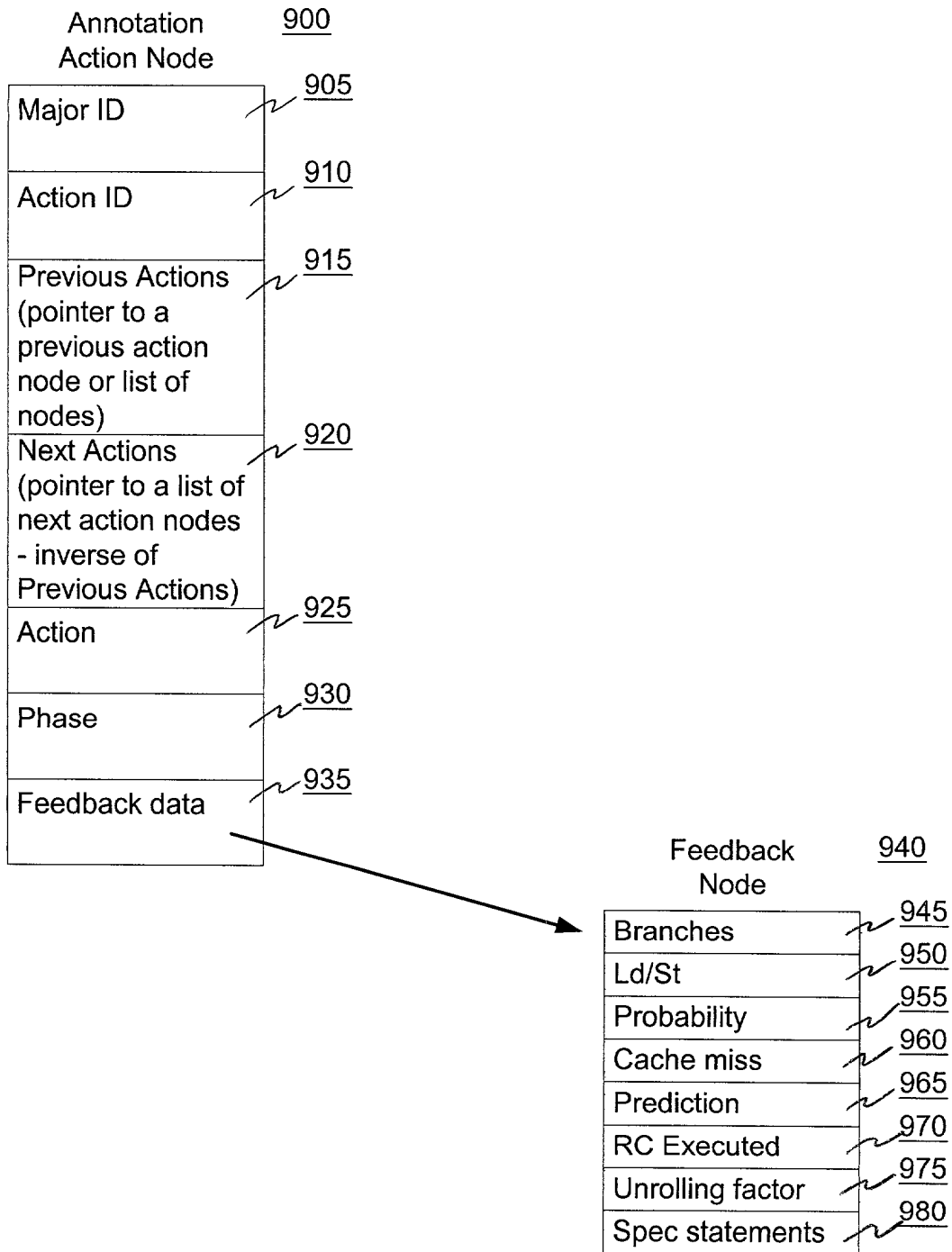


Figure 9

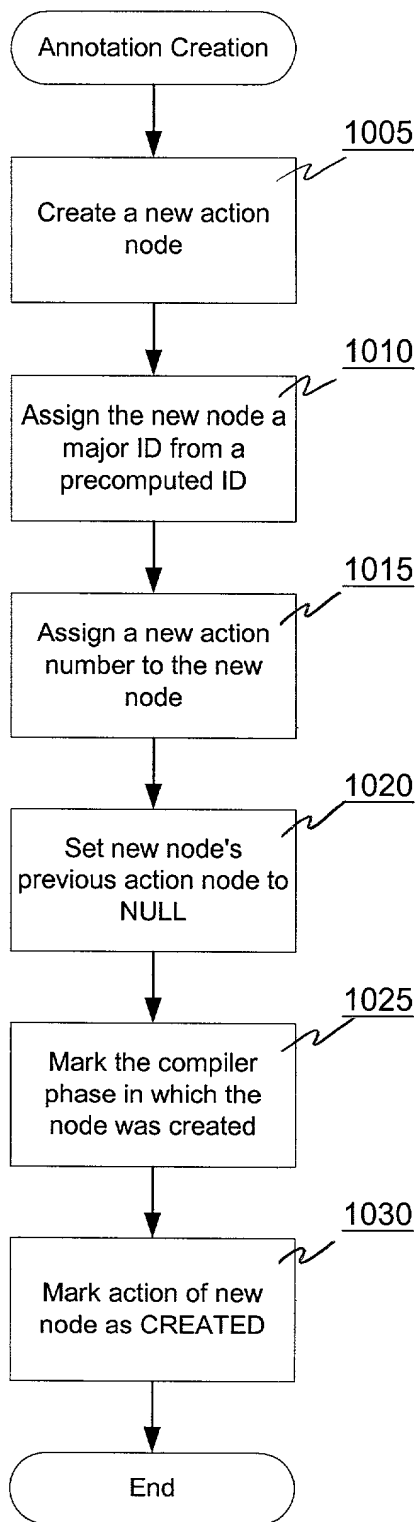


Figure 10

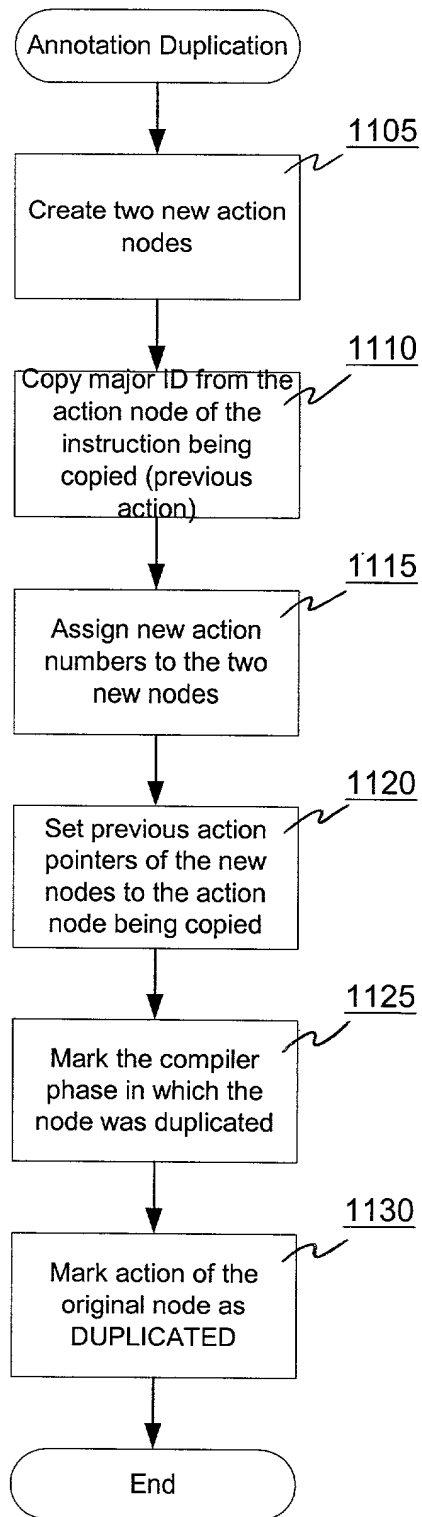


Figure 11

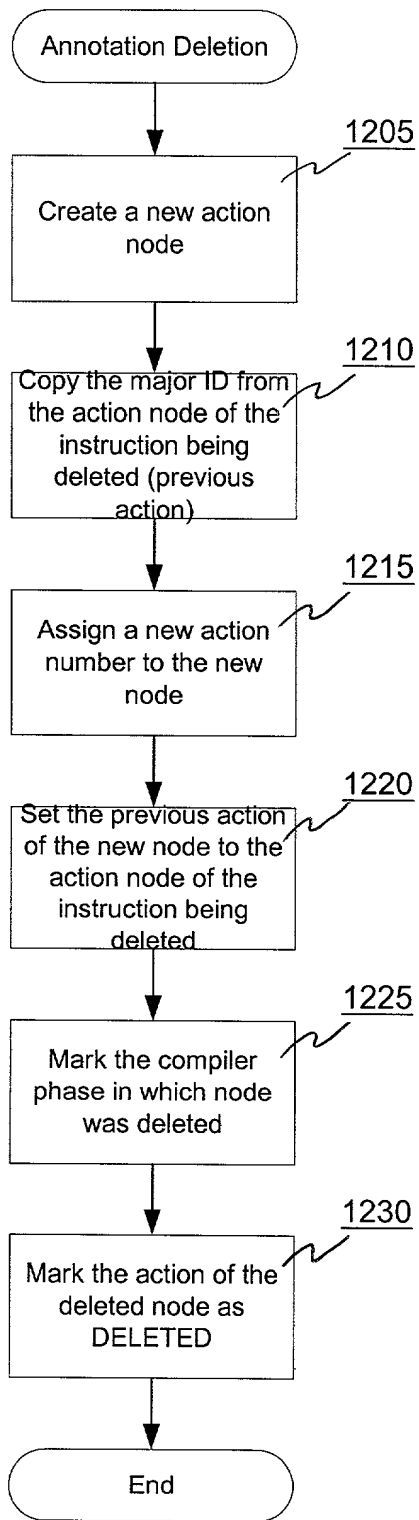


Figure 12

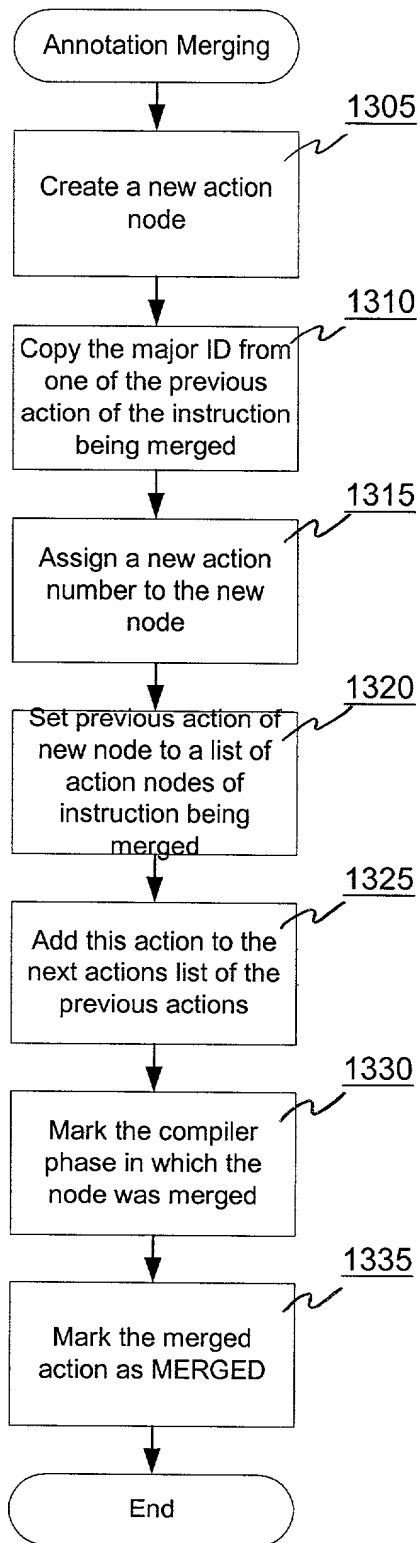


Figure 13

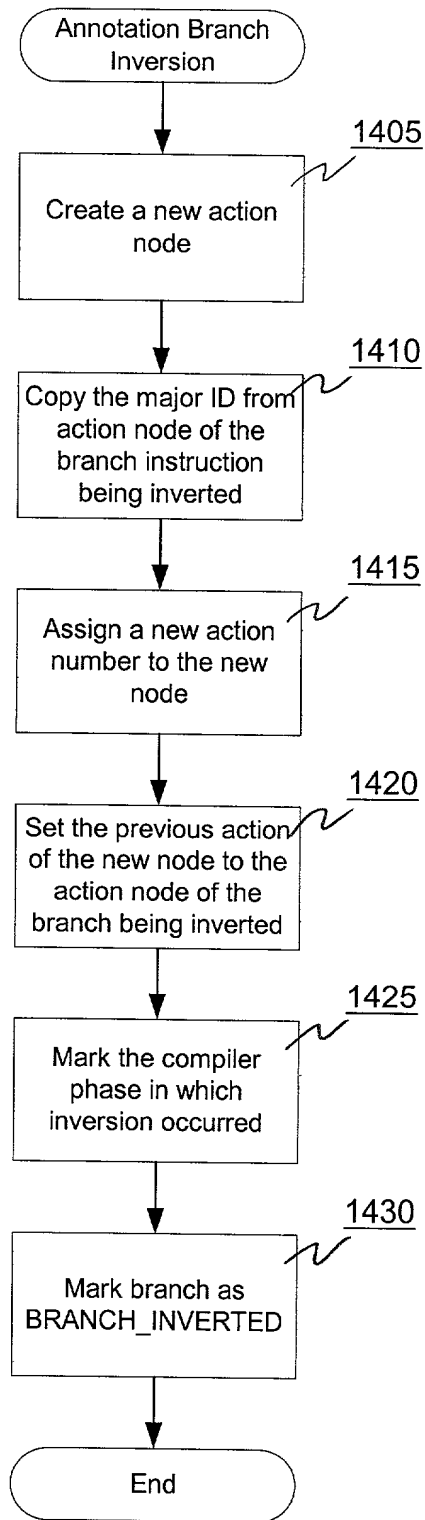


Figure 14

Prior Foreign Application(s)

Priority
Claimed

_____ (Number)	_____ (Country)	_____ (Day/Month/Year Filed)	<u>Yes</u>	<u>No</u>
_____ (Number)	_____ (Country)	_____ (Day/Month/Year Filed)	<u>Yes</u>	<u>No</u>
_____ (Number)	_____ (Country)	_____ (Day/Month/Year Filed)	<u>Yes</u>	<u>No</u>

I hereby claim the benefit under Title 35, United States Code, Section 119(e) of any United States provisional application(s) listed below:

_____ Application Number	_____ Filing Date
_____ Application Number	_____ Filing Date

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose all information known to me to be material to patentability as defined in Title 37, Code of Federal Regulations, Section 1.56 which became available between the filing date of the prior application and the national or PCT international filing date of this application:

_____ Application Number	_____ Filing Date	_____ Status -- patented, pending, abandoned
_____ Application Number	_____ Filing Date	_____ Status -- patented, pending, abandoned

I hereby appoint the persons listed on Appendix A hereto (which is incorporated by reference and a part of this document) as my respective patent attorneys and patent agents, with full power of substitution and revocation, to prosecute this application and to transact all business in the Patent and Trademark Office connected herewith.

Send correspondence to Michael A. DeSanctis, BLAKELY, SOKOLOFF, TAYLOR &
(Name of Attorney or Agent)

ZAFMAN LLP, 12400 Wilshire Boulevard 7th Floor, Los Angeles, California 90025 and direct
telephone calls to Michael A. DeSanctis, (408) 720-8300.
(Name of Attorney or Agent)

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Sole/First Inventor Jayashankar Bharadwaj

Inventor's Signature _____ Date _____

Residence Saratoga, CA _____ Citizenship U.S.A. _____
(City, State) (Country)

Post Office Address 19553 Brockton Lane
Saratoga, California 95070

Full Name of Second/Joint Inventor Ravi Narayanaswamy

Inventor's Signature _____ Date _____

Residence San Jose, CA _____ Citizenship U.S.A. _____
(City, State) (Country)

Post Office Address 1336 Muench Ct.
San Jose, California 95131

Full Name of Third/Joint Inventor _____

Inventor's Signature _____ Date _____

Residence _____ Citizenship _____
(City, State) (Country)

Post Office Address _____

APPENDIX A

William E. Alford, Reg. No. 37,764; Farzad E. Amini, Reg. No. P42,261; Aloysius T. C. AuYeung, Reg. No. 35,432; William Thomas Babbitt, Reg. No. 39,591; Carol F. Barry, Reg. No. 41,600; Jordan Michael Becker, Reg. No. 39,602; Lisa N. Benado, Reg. No. 39,995; Bradley J. Bereznak, Reg. No. 33,474; Michael A. Bernadicou, Reg. No. 35,934; Roger W. Blakely, Jr., Reg. No. 25,831; R. Alan Burnett, Reg. No. 46,149; Gregory D. Caldwell, Reg. No. 39,926; Andrew C. Chen, Reg. No. 43,544; Thomas M. Coester, Reg. No. 39,637; Donna Jo Coningsby, Reg. No. 41,684; Florin Corie, Reg. No. 46,244; Dennis M. deGuzman, Reg. No. 41,702; Stephen M. De Klerk, Reg. No. P46,503; Michael Anthony DeSanctis, Reg. No. 39,957; Daniel M. De Vos, Reg. No. 37,813; Robert Andrew Diehl, Reg. No. 40,992; Sanjeet Dutta, Reg. No. P46,145; Matthew C. Fagan, Reg. No. 37,542; Tarek N. Fahmi, Reg. No. 41,402; George Fountain, Reg. No. 37,374; Paramita Ghosh, Reg. No. 42,806; James Y. Go, Reg. No. 40,621; Libby N. Ho, Reg. No. P46,774; James A. Henry, Reg. No. 41,064; Willmore F. Holbrow III, Reg. No. P41,845; Sheryl Sue Holloway, Reg. No. 37,850; George W. Hoover II, Reg. No. 32,992; Eric S. Hyman, Reg. No. 30,139; William W. Kidd, Reg. No. 31,772; Sang Hui Kim, Reg. No. 40,450; Walter T. Kim, Reg. No. 42,731; Eric T. King, Reg. No. 44,188; Erica W. Kuo, Reg. No. 42,775; George Brian Leavell, Reg. No. 45,436; Kurt P. Leyendecker, Reg. No. 42,799; Gordon R. Lindeen III, Reg. No. 33,192; Jan Carol Little, Reg. No. 41,181; Joseph Lutz, Reg. No. 43,765; Michael J. Mallie, Reg. No. 36,591; Andre L. Marais, under 37 C.F.R. § 10.9(b); Paul A. Mendonsa, Reg. No. 42,879; Clive D. Menezes, Reg. No. 45,493; Chun M. Ng, Reg. No. 36,878; Thien T. Nguyen, Reg. No. 43,835; Thinh V. Nguyen, Reg. No. 42,034; Dennis A. Nicholls, Reg. No. 42,036; Daniel E. Ovanezian, Reg. No. 41,236; Kenneth B. Paley, Reg. No. 38,989; Marina Portnova, Reg. No. P45,750; William F. Ryann, Reg. No. 44,313; James H. Salter, Reg. No. 35,668; William W. Schaal, Reg. No. 39,018; James C. Scheller, Reg. No. 31,195; Jeffrey Sam Smith, Reg. No. 39,377; Maria McCormack Sobrino, Reg. No. 31,639; Stanley W. Sokoloff, Reg. No. 25,128; Judith A. Szepesi, Reg. No. 39,393; Vincent P. Tassinari, Reg. No. 42,179; Edwin H. Taylor, Reg. No. 25,129; John F. Travis, Reg. No. 43,203; Joseph A. Twarowski, Reg. No. 42,191; Tom Van Zandt, Reg. No. 43,219; Lester J. Vincent, Reg. No. 31,460; Glenn E. Von Tersch, Reg. No. 41,364; John Patrick Ward, Reg. No. 40,216; Mark L. Watson, Reg. No. P46,322; Thomas C. Webster, Reg. No. P46,154; Steven D. Yates, Reg. No. 42,242; and Norman Zafman, Reg. No. 26,250; my patent attorneys, and Firasat Ali, Reg. No. 45,715; and Justin M. Dillon, Reg. No. 42,486; my patent agents, of BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP, with offices located at 12400 Wilshire Boulevard, 7th Floor, Los Angeles, California 90025, telephone (310) 207-3800, and Alan K. Aldous, Reg. No. 31,905; Edward R. Brake, Reg. No. 37,784; Ben Burge, Reg. No. 42,372; Jeffrey S. Draeger, Reg. No. 41,000; Cynthia Thomas Faatz, Reg. No. 39,973; John N. Greaves, Reg. No. 40,362; Seth Z. Kalson, Reg. No. 40,670; David J. Kaplan, Reg. No. 41,105; Peter Lam, Reg. No. 44,855; Charles A. Mirho, Reg. No. 41,199; Leo V. Novakoski, Reg. No. 37,198; Thomas C. Reynolds, Reg. No. 32,488; Kenneth M. Seddon, Reg. No. 43,105; Mark Seeley, Reg. No. 32,299; Steven P. Skabrat, Reg. No. 36,279; Howard A. Skaist, Reg. No. 36,008; Gene I. Su, Reg. No. 45,140; Calvin E. Wells, Reg. No. P43,256; Raymond J. Werner, Reg. No. 34,752; Robert G. Winkle, Reg. No. 37,474; and Charles K. Young, Reg. No. 39,435; my patent attorneys, of INTEL CORPORATION; and James R. Thein, Reg. No. 31,710, my patent attorney with full power of substitution and revocation, to prosecute this application and to transact all business in the Patent and Trademark Office connected herewith.

APPENDIX B

Title 37, Code of Federal Regulations, Section 1.56 Duty to Disclose Information Material to Patentability

(a) A patent by its very nature is affected with a public interest. The public interest is best served, and the most effective patent examination occurs when, at the time an application is being examined, the Office is aware of and evaluates the teachings of all information material to patentability. Each individual associated with the filing and prosecution of a patent application has a duty of candor and good faith in dealing with the Office, which includes a duty to disclose to the Office all information known to that individual to be material to patentability as defined in this section. The duty to disclosure information exists with respect to each pending claim until the claim is cancelled or withdrawn from consideration, or the application becomes abandoned. Information material to the patentability of a claim that is cancelled or withdrawn from consideration need not be submitted if the information is not material to the patentability of any claim remaining under consideration in the application. There is no duty to submit information which is not material to the patentability of any existing claim. The duty to disclose all information known to be material to patentability is deemed to be satisfied if all information known to be material to patentability of any claim issued in a patent was cited by the Office or submitted to the Office in the manner prescribed by §§1.97(b)-(d) and 1.98. However, no patent will be granted on an application in connection with which fraud on the Office was practiced or attempted or the duty of disclosure was violated through bad faith or intentional misconduct. The Office encourages applicants to carefully examine:

- (1) Prior art cited in search reports of a foreign patent office in a counterpart application, and
 - (2) The closest information over which individuals associated with the filing or prosecution of a patent application believe any pending claim patentably defines, to make sure that any material information contained therein is disclosed to the Office.
- (b) Under this section, information is material to patentability when it is not cumulative to information already of record or being made of record in the application, and
- (1) It establishes, by itself or in combination with other information, a prima facie case of unpatentability of a claim; or
 - (2) It refutes, or is inconsistent with, a position the applicant takes in:
 - (i) Opposing an argument of unpatentability relied on by the Office, or
 - (ii) Asserting an argument of patentability.

A prima facie case of unpatentability is established when the information compels a conclusion that a claim is unpatentable under the preponderance of evidence, burden-of-proof standard, giving each term in the claim its broadest reasonable construction consistent with the specification, and before any consideration is given to evidence which may be submitted in an attempt to establish a contrary conclusion of patentability.

- (c) Individuals associated with the filing or prosecution of a patent application within the meaning of this section are:
- (1) Each inventor named in the application;
 - (2) Each attorney or agent who prepares or prosecutes the application; and
 - (3) Every other person who is substantively involved in the preparation or prosecution of the application and who is associated with the inventor, with the assignee or with anyone to whom there is an obligation to assign the application.
- (d) Individuals other than the attorney, agent or inventor may comply with this section by disclosing information to the attorney, agent, or inventor.